

Steven M. Hoffberg

From: Steven M. Hoffberg [steve@hoffberg.org]
Sent: Thursday, November 18, 2004 12:20 PM
To: 'Nguyen, Nga'
Subject: 09/599,163 profile.c

```

/* -----
 * -----
 * Helper functions for user profile data structure
 *
 * (c) Copyright 1995 Newshare Corporation
 * -----
 */

#define PROFILE_IMPLEMENTATION                /* needs to be at top */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <bstring.h>
#include <sys/time.h>
#include <sys/types.h>
#include <netinet/in.h>
#ifdef SOLARIS2
#define bcopy(b1,b2,len) memmove(b2, b1, (size_t)(len))
#define bzero(b,len) memset(b, 0, (size_t)(len))
#define bcmp(b1,b2,len) memcmp(b1, b2, (size_t)(len))
#endif /* SOLARIS2 */

#include "coding.h"

#define PUBLIC
#define PRIVATE static
#define MALLOC(c) malloc(c)

/* User Profile Data Structures */

/* the service class */

struct service_class {
    unsigned int class_id: 4;           /* class identification */
    unsigned int pgn_limit: 4;          /* page count delivery limit */
    unsigned int priority: 4;           /* service priority */
    unsigned int pgcl_limit: 2;         /* corresponds to FLAG bits in page-class */
    unsigned int unused1: 2;
    unsigned int unused2: 16;
};

```

```

/* encoded user preferences */

struct user_preferences {
    /* "sensitivity" flags */

    unsigned int pdac_flag: 1; /* parent discretion / adult content limit */
    unsigned int privacy1_flag: 1; /* yes/no corrolate user to use */
    unsigned int prem_chg_flag: 1; /* do not notify for hi-cost pages */
    unsigned int unusedf4: 1;
    unsigned int unusedf5: 1;
    unsigned int unusedf6: 1;
    unsigned int adv_ctxt: 2; /* advertising context */

    /* additional preferences */

    unsigned int cust_grp: 4; /* customer group */
    unsigned int unused7: 4;
    unsigned int unused8: 16;
};

/* profile of a user */
/* NOTE: all (unsigned long) in net byte order. h_id stays that way */

struct user_profile {
    unsigned long h_id; /* IP address of current host */
    unsigned long u_id; /* global newshare user identifier */
    unsigned long p_id; /* global ID of this user's PM */
    unsigned long ses_id; /* session ID of the current session */
    struct service_class sc;
    struct user_preferences up;
};

typedef struct user_profile *TVS_PROFILE;

/*
 * legitimate way to treat the profile as bytes :-)
 */

union up_bytes {
    struct user_profile up;
    char upc[sizeof(struct user_profile)];
    int upi[(sizeof(struct user_profile) / sizeof(unsigned long))];
};

/* magic padding bytes */

#define MAGIC_UP_PAD 0x0f0f0
#define MAGIC_SC_PAD 0x0f0f0

#include "tvs_profile.h"

```

```

/* -----
 * exported functions to get info in/out of the profile
 * -----
 */

PUBLIC int
tvs_profile_is_valid(TVS_PROFILE prof)
{
    if ((prof->up.unused8 == MAGIC_UP_PAD) && (prof->sc.unused2 == MAGIC_SC_PAD))
        return 1;
    else
        return 0;
}

PUBLIC int
tvs_sizeof_profile()
{
    return sizeof(struct user_profile);
}

PUBLIC TVS_PROFILE
tvs_make_user_profile()
{
    TVS_PROFILE prof;

    prof = (TVS_PROFILE) MALLOC(sizeof(struct user_profile));
    if (!prof) return (TVS_PROFILE) NULL;

    bzero((char *)prof, sizeof(struct user_profile));
    prof->up.unused8 = MAGIC_UP_PAD;
    prof->sc.unused2 = MAGIC_SC_PAD;

    /* setup default values */

    tvs_set_service_class(prof, DEFAULT_SERVICE_CLASS);
    tvs_set_page_class_limit(prof, DEFAULT_PAGE_CLASS_LIMIT);
    tvs_set_service_priority(prof, DEFAULT_SERVICE_PRIORITY);
    tvs_set_customer_group(prof, DEFAULT_CUSTOMER_GROUP);
    tvs_set_adv_context(prof, DEFAULT_ADV_LEVEL);
    tvs_set_pdac_flag(prof, DEFAULT_PDAC_FLAG);
    tvs_set_privacy1_flag(prof, DEFAULT_PRIVACY_FLAG);
    tvs_set_premium_flag(prof, DEFAULT_PREMIUM_FLAG);

    return prof;
}

PUBLIC TVS_PROFILE
tvs_make_testdrive_profile()
{

```

```

TVS_PROFILE prof;

prof = (TVS_PROFILE) MALLOC(sizeof(struct user_profile));
if (!prof) return (TVS_PROFILE) NULL;

bzero((char *)prof, sizeof(struct user_profile));
prof->up.unused8 = MAGIC_UP_PAD;
prof->sc.unused2 = MAGIC_SC_PAD;

/* set special testdrive values */

tvs_set_service_class(prof, TESTDRIVE_SERVICE_CLASS);
tvs_set_page_class_limit(prof, TESTDRIVE_PAGE_CLASS_LIMIT);
tvs_set_service_priority(prof, TESTDRIVE_SERVICE_PRIORITY);
tvs_set_customer_group(prof, TESTDRIVE_CUSTOMER_GROUP);
tvs_set_adv_context(prof, TESTDRIVE_ADV_LEVEL);
tvs_set_pdac_flag(prof, TESTDRIVE_PDAC_FLAG);
tvs_set_privacy1_flag(prof, TESTDRIVE_PRIVACY_FLAG);
tvs_set_premium_flag(prof, TESTDRIVE_PREMIUM_FLAG);

return prof;
}

PUBLIC void
tvs_set_userid(TVS_PROFILE prof, unsigned long uid)
{
    if (prof)
        prof->u_id = uid;
}

PUBLIC void
tvs_set_pmid(TVS_PROFILE prof, unsigned long pmid)
{
    if (prof)
        prof->p_id = pmid;
}

PUBLIC void
tvs_set_hostid(TVS_PROFILE prof, unsigned long hostid)
{
    if (prof)
        prof->h_id = hostid; /* already in net byte order from gethostbyname() */
}

#if 0
PUBLIC unsigned long
tvs_make_sessionid(unsigned long pm_id, unsigned long id)
{
    unsigned long sid;

```

```

    sid = ((pm_id & 0xffff) << 16) + (id & 0xffff);
    return sid;
}
#else
PUBLIC unsigned long
tvs_make_sessionid(unsigned long pm_id, unsigned long id)
{
    struct timeval tv;
    unsigned int tmp;

#define BIT_MSK    0xff
#define UNBIT_MSK 8

    if (gettimeofday(&tv, NULL) == 0) {
        tmp = (tv.tv_sec << UNBIT_MSK) | ((tv.tv_usec >> 4) & BIT_MSK);
        return tmp;
    }
    else
        return 0x11223344;
}
#endif
PUBLIC void
tvs_set_sessionid(TVS_PROFILE prof, unsigned long sid)
{
    if (prof)
        prof->ses_id = sid;
}

PUBLIC void
tvs_set_service_class(TVS_PROFILE prof, int class)
{
    if (prof)
        prof->sc.class_id = class;
}

PUBLIC void
tvs_set_page_class_limit(TVS_PROFILE prof, int limit)
{
    if (prof)
        prof->sc.pgcl_limit = limit;
}

PUBLIC void
tvs_set_page_count_limit(TVS_PROFILE prof, int count)
{
    if (prof)
        prof->sc.pgn_limit = count;
}

PUBLIC void
tvs_set_service_priority(TVS_PROFILE prof, int priority)

```

```

{
    if (prof)
        prof->sc.priority = priority;
}

PUBLIC void
tvs_set_customer_group(TVS_PROFILE prof, int group)
{
    if (prof)
        prof->up.cust_grp = group;
}

PUBLIC void
tvs_set_adv_context(TVS_PROFILE prof, int ac)
{
    if (prof)
        prof->up.adv_ctxt = ac;
}

PUBLIC void
tvs_set_pdac_flag(TVS_PROFILE prof, int flag)
{
    if (prof)
        prof->up.pdac_flag = flag;
}

PUBLIC void
tvs_set_privacy1_flag(TVS_PROFILE prof, int flag)
{
    if (prof)
        prof->up.privacy1_flag = flag;
}

PUBLIC void
tvs_set_premium_flag(TVS_PROFILE prof, int flag)
{
    if (prof)
        prof->up.prem_chg_flag = flag;
}

PUBLIC unsigned long
tvs_get_userid(TVS_PROFILE prof)
{
    return prof->u_id;
}

PUBLIC unsigned long
tvs_get_pmid(TVS_PROFILE prof)
{
    return prof->p_id;
}

```

```
PUBLIC unsigned long  
tvs_get_hostid(TVS_PROFILE prof)  
{  
    return prof->h_id; /* leave in net byte order */  
}
```

```
PUBLIC unsigned long  
tvs_get_sessionid(TVS_PROFILE prof)  
{  
    return prof->ses_id;  
}
```

```
PUBLIC int  
tvs_get_service_class(TVS_PROFILE prof)  
{  
    return (int) prof->sc.class_id;  
}
```

```
PUBLIC int  
tvs_get_page_class_limit(TVS_PROFILE prof)  
{  
    return (int) prof->sc.pgcl_limit;  
}
```

```
PUBLIC int  
tvs_get_page_count_limit(TVS_PROFILE prof)  
{  
    return (int) prof->sc.pgn_limit;  
}
```

```
PUBLIC int  
tvs_get_service_priority(TVS_PROFILE prof)  
{  
    return (int) prof->sc.priority;  
}
```

```
PUBLIC int  
tvs_get_customer_group(TVS_PROFILE prof)  
{  
    return (int) prof->up.cust_grp;  
}
```

```
PUBLIC int  
tvs_get_adv_context(TVS_PROFILE prof)  
{  
    return (int) prof->up.adv_ctxt;  
}
```

```
PUBLIC int  
tvs_get_pdac_flag(TVS_PROFILE prof)
```

```

{
    return (int) prof->up.pdac_flag;
}

PUBLIC int
tvs_get_privacy1_flag(TVS_PROFILE prof)
{
    return (int) prof->up.privacy1_flag;
}

PUBLIC int
tvs_get_premium_flag(TVS_PROFILE prof)
{
    return (int) prof->up.prem_chg_flag;
}

#if 0
/* -----
 * encode a TVS_PROFILE for transmission over the net
 * REPLACED BELOW!!!!
 * -----
 */

PUBLIC char *
tvs_encode_profile(TVS_PROFILE prof)
{
    char *string;
    int len = 128, plen, i;
    union up_bytes *up = (union up_bytes *)prof;

    /* net compatible byte swap */

    plen = tvs_sizeof_profile();
    for (i = 0; i < (plen / sizeof(unsigned long)); i++)
        up->upi[i] = htonl(up->upi[i]);

    string = (char *) MALLOC(len);
    if (!string) return (char *) NULL;

    if (!tvs_encode((char *) prof, tvs_sizeof_profile(), string, &len))
        return (char *) NULL;
    else {
        string[len] = '\0';
        string = realloc(string, strlen(string)+1);
        return (char *)string;
    }
}

/* -----
 * decode an encoded TVS_PROFILE
 * -----

```



```

*/

PUBLIC TVS_PROFILE
tv_decode_profile(char *string)
{
    int len, i;
    TVS_PROFILE up;
    union up_bytes *ub;

    len = tvs_sizeof_profile();
    up = tvs_make_user_profile();
    ub = (union up_bytes *) up;

    if (!tv_decode(string, (char *) up, &len))
        return (TVS_PROFILE) NULL;
    else { /* net compatible byte swap */
        len = tvs_sizeof_profile();
        for (i = 0; i < (len / sizeof(unsigned long)); i++)
            ub->upi[i] = ntohl(ub->upi[i]);
        return up;
    }
}
#endif

/* -----
 * show contents of a profile
 * -----
 */

PUBLIC void
tv_show_user_profile(TVS_PROFILE prof)
{
    if (!prof) return;

    printf("profile:\nhost id:\t0x%lx\nuser id:\t0x%lx\n",
        tvs_get_hostid(prof), tvs_get_userid(prof));

    printf("pm id:\t0x%lx\n", tvs_get_pmid(prof));

    printf("session:\t0x%lx\n", tvs_get_sessionid(prof));

    printf("service class:\t0x%lx\n", tvs_get_service_class(prof));
    printf("page cnt limit:\t0x%lx\n", tvs_get_page_count_limit(prof));
    printf("serv pr:\t0x%lx\n", tvs_get_service_priority(prof));
    printf("page class lim:\t0x%lx\n", tvs_get_page_class_limit(prof));

    printf("pdac flag:\t0x%lx\n", tvs_get_pdac_flag(prof));
    printf("privacy1:\t0x%lx\n", tvs_get_privacy1_flag(prof));
    printf("prem chg:\t0x%lx\n", tvs_get_premium_flag(prof));

    printf("adv ctx:\t0x%lx\nc group:\t0x%lx\n",

```

```

        tvs_get_adv_context(prof), tvs_get_customer_group(prof));

    return;
}

/* -----
 * make the flag chars that go into the log file
 * -----
 */

PUBLIC char *
tvs_make_preference_flags(TVS_PROFILE prof)
{
    static char flags[11];
    unsigned int fbits = 0;

    memcpy(&fbits, &(prof->up), 4);

    sprintf(flags, "0x%08x", fbits);
    return flags;
}

/* -----
 * endian neutral, quick bit field coding for the profile
 * -----
 */

static int hexcodes[16] =
{0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x61,0x62,0x63,0x64,0x65,0x66};

PUBLIC char *
tvs_encode_profile(TVS_PROFILE prof)
{
    char *buffer;
    int i = 0, j = 0;

    buffer = (char *) malloc(128);
    if (!buffer) return (char *) NULL;

    sprintf(buffer, "%08lx", ntohl(prof->h_id));
    sprintf(&buffer[i+=8], "%08lx", prof->u_id);
    sprintf(&buffer[i+=8], "%08lx", prof->p_id);
    sprintf(&buffer[i+=8], "%08lx", prof->ses_id);

    /* service class */
    i += 8;
    buffer[i++] = hexcodes[prof->sc.class_id & 0xf];
    buffer[i++] = hexcodes[prof->sc.pgn_limit & 0xf];
    buffer[i++] = hexcodes[prof->sc.priority & 0xf];
    j = ((prof->sc.pgcl_limit & 0x3) << 2) + prof->sc.unused1;
    buffer[i++] = hexcodes[j];
    buffer[i++] = hexcodes[(prof->sc.unused2 >> 12) & 0xf];

```

```

buffer[i++] = hexcodes[(prof->sc.unused2 >> 8) & 0xf];
buffer[i++] = hexcodes[(prof->sc.unused2 >> 4) & 0xf];
buffer[i++] = hexcodes[(prof->sc.unused2 & 0xf)];

/* user preferences */

j |= prof->up.pdac_flag;
j = (j << 1) | (prof->up.privacy1_flag);
j = (j << 1) | (prof->up.prem_chg_flag);
j = (j << 1) | (prof->up.unusedf4 & 0x1);

buffer[i++] = hexcodes[(j & 0xf)];
j = prof->up.unusedf5 & 0x1;
j = (j << 1) | (prof->up.unusedf6);
j = (j << 2) | (prof->up.adv_ctxt & 0x3);

buffer[i++] = hexcodes[j];
buffer[i++] = hexcodes[(prof->up.cust_grp & 0xf)];

j = prof->up.unused7;
buffer[i++] = hexcodes[(j & 0xf)];
j = prof->up.unused8;
buffer[i++] = hexcodes[(j >> 12) & 0xf];
buffer[i++] = hexcodes[(j >> 8) & 0xf];
buffer[i++] = hexcodes[(j >> 4) & 0xf];
buffer[i++] = hexcodes[(j & 0xf)];

buffer[i] = '\0';
return buffer;
}

```

```
#define HEXVAL(c) ((c) > '9' ? (c) - 'a' + 10 : (c) - '0')
```

```

PUBLIC TVS_PROFILE
tvs_decode_profile(char *buffer)
{
    TVS_PROFILE prof;
    int i = 0;
    unsigned int k, l, m, n;
    unsigned long j;

    prof = tvs_make_user_profile();
    if (!prof) return prof;

    sscanf(buffer, "%8lx", &(prof->h_id));
    prof->h_id = htonl(prof->h_id);
    sscanf(&buffer[i+=8], "%8lx", &(prof->u_id));
    sscanf(&buffer[i+=8], "%8lx", &(prof->p_id));
    sscanf(&buffer[i+=8], "%8lx", &(prof->ses_id));

    /* service class */

```

```

i += 8;
prof->sc.class_id = HEXVAL(buffer[i]); i++;
prof->sc.pgn_limit = HEXVAL(buffer[i]); i++;
prof->sc.priority = HEXVAL(buffer[i]); i++;

k = HEXVAL(buffer[i]); i++;
prof->sc.pgcl_limit = (k >> 2) & 0x3;
prof->sc.unused1 = (k & 0x3);

k = HEXVAL(buffer[i]); i++;
l = HEXVAL(buffer[i]); i++;
m = HEXVAL(buffer[i]); i++;
n = HEXVAL(buffer[i]); i++;

j = ((k & 0xf) << 12) |
    ((l & 0xf) << 8) |
    ((m & 0xf) << 4) |
    (n & 0xf);

prof->sc.unused2 = j & 0xffff;

/* user preferences */

n = HEXVAL(buffer[i]); i++;
prof->up.pdac_flag = (n >> 3) & 0x1;
prof->up.privacy1_flag = (n >> 2) & 0x1;
prof->up.prem_chg_flag = (n >> 1) & 0x1;
prof->up.unusedf4 = n & 0x1;

n = HEXVAL(buffer[i]); i++;

prof->up.unusedf5 = (n >> 3) & 0x1;
prof->up.unusedf6 = (n >> 2) & 0x1;
prof->up.adv_ctxt = n & 0x3;

prof->up.cust_grp = HEXVAL(buffer[i]); i++;

prof->up.unused7 = HEXVAL(buffer[i]); i++;

k = HEXVAL(buffer[i]); i++;
l = HEXVAL(buffer[i]); i++;
m = HEXVAL(buffer[i]); i++;
n = HEXVAL(buffer[i]); i++;

prof->up.unused8 = ((k & 0xf) << 12) | ((l & 0xf) << 8) | ((m & 0xf) << 4) | (n & 0xf);

return prof;
}

```

Very truly yours,

Steven M. Hoffberg
Milde & Hoffberg, LLP
Suite 460
10 Bank Street
White Plains, NY 10606
(914) 949-3100 tel.
(914) 949-3416 fax
steve@hoffberg.org
www.hoffberg.org

Confidentiality Notice: This message, and any attachments thereto, may contain confidential information which is legally privileged. The information is intended only for the use of the intended recipient, generally the individual or entity named above. If you believe you are not the intended recipient, or in the event that this document is received in error, or misdirected, you are requested to immediately inform the sender by reply e-mail at Steve@Hoffberg.org and destroy all copies of the e-mail file and attachments. You are hereby notified that any disclosure, copying, distribution or use of any information contained in this transmission other than by the intended recipient is strictly prohibited.